K Means Clustering Algorithm

SHUBHAM PATEL

BATCH - 03

SERIAL NO. 104

Contents

What is clustering?

What is K Means Clustering Algorithm?

Algorithm/Working of K means clustering algorithm

Sklearn.cluster.KMeans()

Implementation of K Means clustering algorithm

Applications of K means clustering algorithm

Advantages and disadvantages of K means clustering

Clustering

An unsupervised learning method.

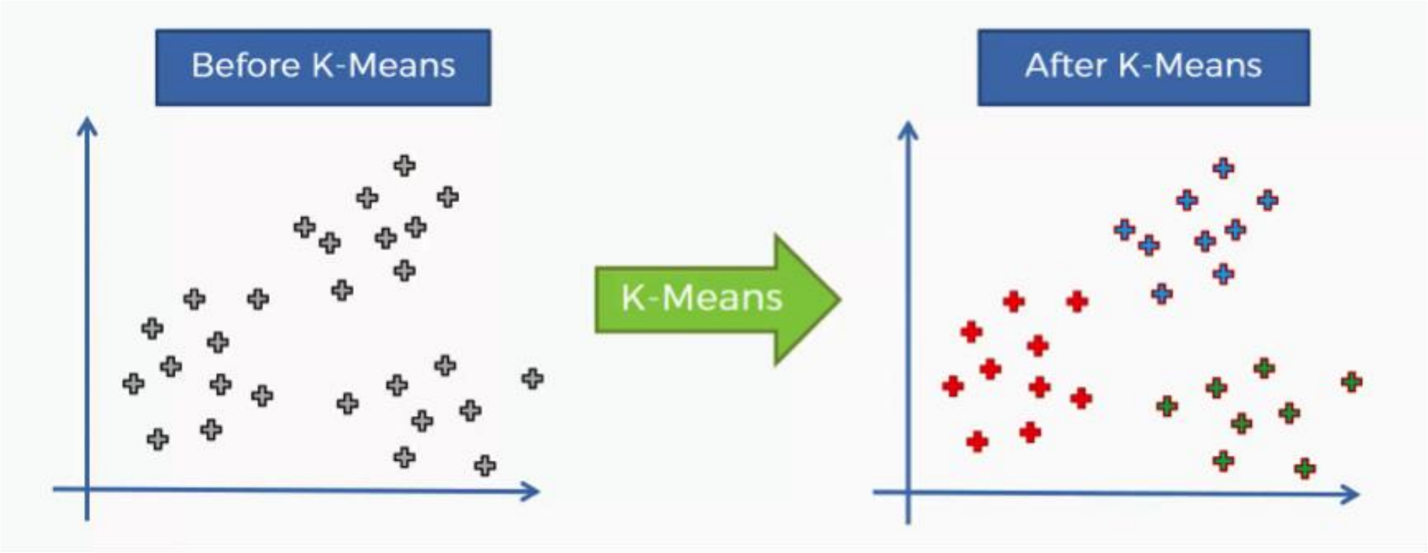
No predefined labels to cluster objects.

Types of Clustering:

- K Means clustering
- K Mediods clustering
- DBSCAN (Density bases special clustering of applications and noise)
- OPTICS (Ordering Points to identify Clustering structure)
- Hierarchical Clustering

K Means Clustering

K-means is a clustering algorithm which is centroid-based or we can say is based on the center point of each cluster, along with distance-based or we can say the distance between the cluster center explained above and points in a cluster , where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is represented by a centroid or the cluster mean.



Algorithm/Working for K Means Clustering

- Step 1: Choose the number of clusters (k) you want.
- Step 2: Start with k centorids by putting them at random place.
- Step 3: Compute distance from every point from centroid and cluster them accordingly.
- Step 4: Adjust centorids such that they become center point of the cluster just formed.
- Step 5: Again re-cluster every point based on their distance with centroid.
- Continue above steps until stopping criteria is not met.
- As soon as stopping criteria is met, stop and the clusters present are final clusters or the results we require.

Stopping Criteria for K Means Clustering Algorithm:

- Centroids of the newly points cluster have'nt changed
- Points in a clusters do not change their location.
- Maximum number of iterations are done.

Few important parameters of sklearn.cluster.KMeans()

n_clusters: int, default=8
The number of clusters to form as well as the number of centroids to generate.

init: {‘k-means++’, ‘random’}, callable or array-like of shape (n_clusters, n_features), default=’k-means++’
Method for initialization

n_init: int, default=10
Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs.

max_iter: int, default=300
Maximum number of iterations of the k-means algorithm for a single run.

random_state: int, RandomState instance or None, default=None
Determines random number generation for centroid initialization.

Few important attributes of sklearn.cluster.KMeans()

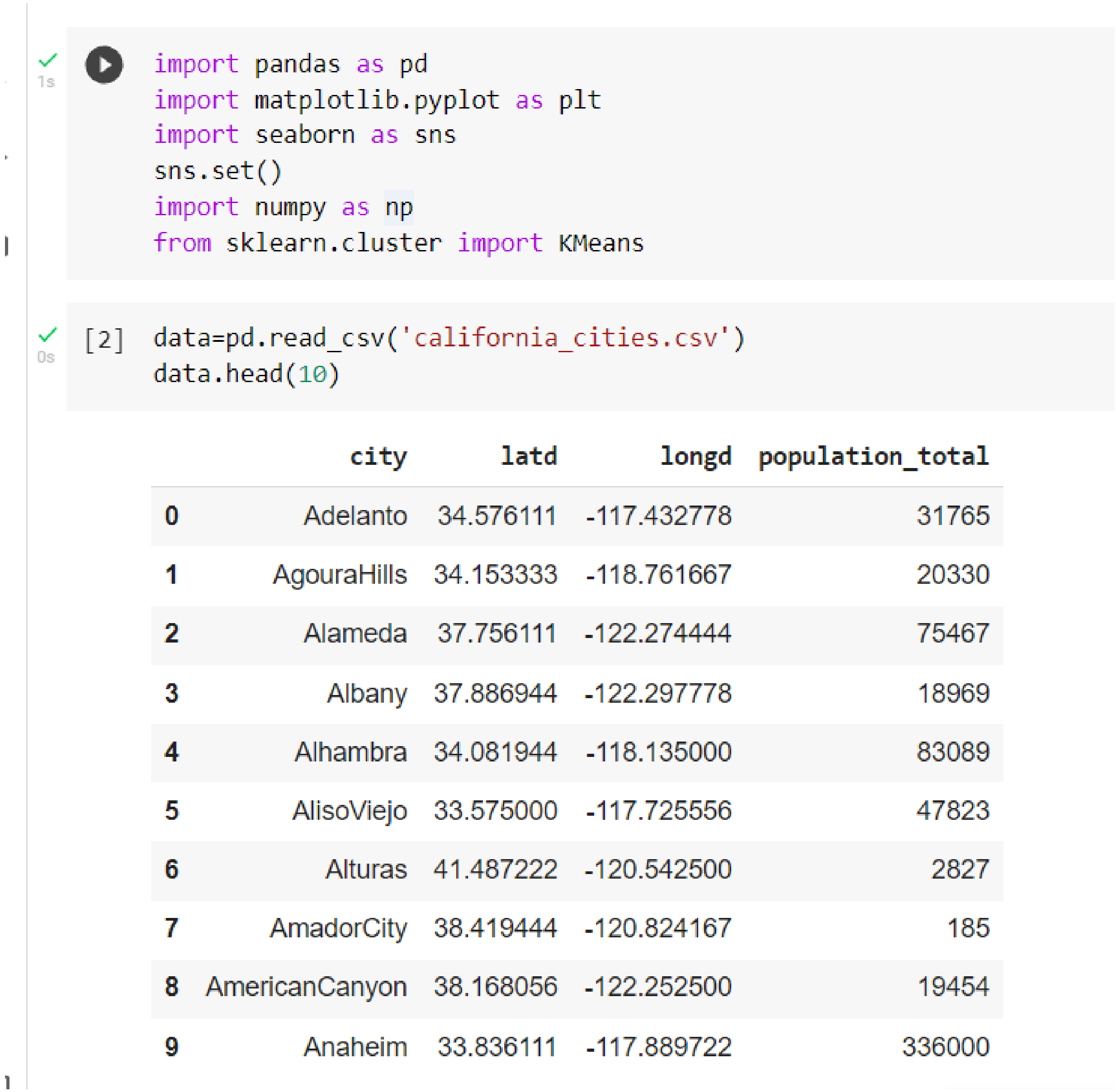
cluster_centers_: ndarray of shape (n_clusters, n_features)
Coordinates of cluster centers

labels_: ndarray of shape (n_samples)
Labels of each point

inertia_: float
Sum of squared distances of samples to their closest cluster center.

n_iter_: int
Number of iterations run.

Implementing K Means clustering Importing required libraries and reading the dataset

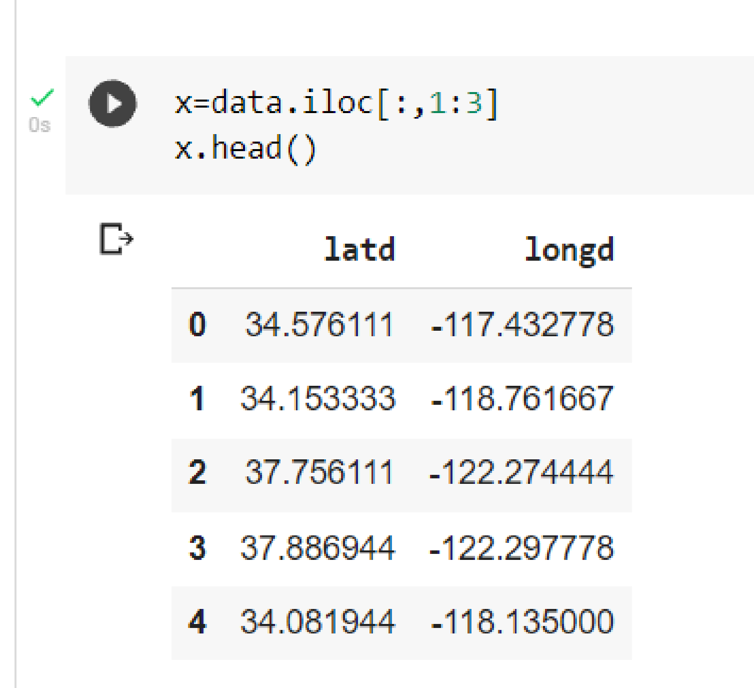


```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import numpy as np
from sklearn.cluster import KMeans
```

```
data=pd.read_csv('california_cities.csv')
data.head(10)
```

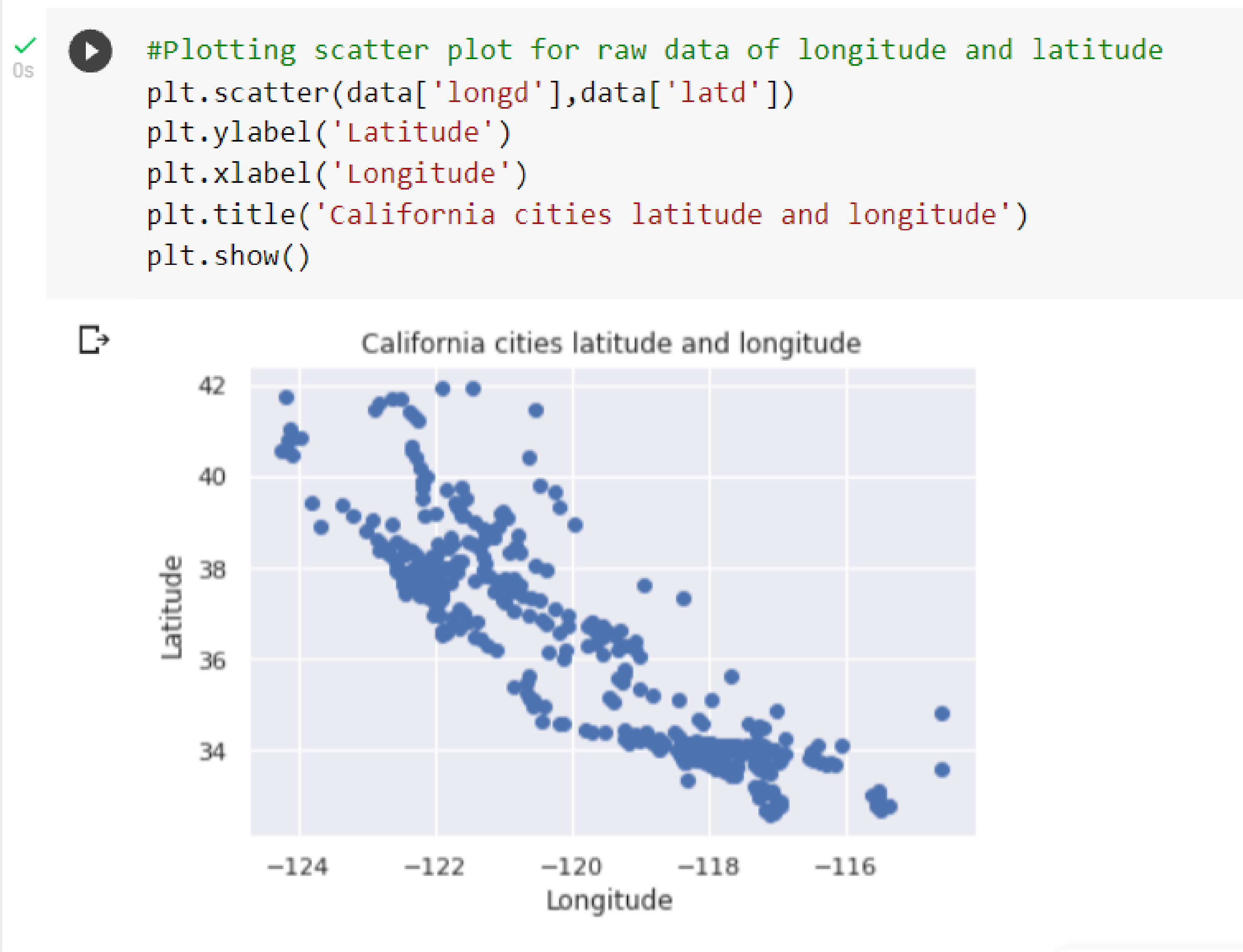
	city	latd	longd	population_total
0	Adelanto	34.576111	-117.432778	31765
1	AgouraHills	34.153333	-118.761667	20330
2	Alameda	37.756111	-122.274444	75467
3	Albany	37.886944	-122.297778	18969
4	Alhambra	34.081944	-118.135000	83089
5	AlisoViejo	33.575000	-117.725556	47823
6	Alturas	41.487222	-120.542500	2827
7	AmadorCity	38.419444	-120.824167	185
8	AmericanCanyon	38.168056	-122.252500	19454
9	Anaheim	33.836111	-117.889722	336000

Unclustered data



```
x=data.iloc[:,1:3]
x.head()
```

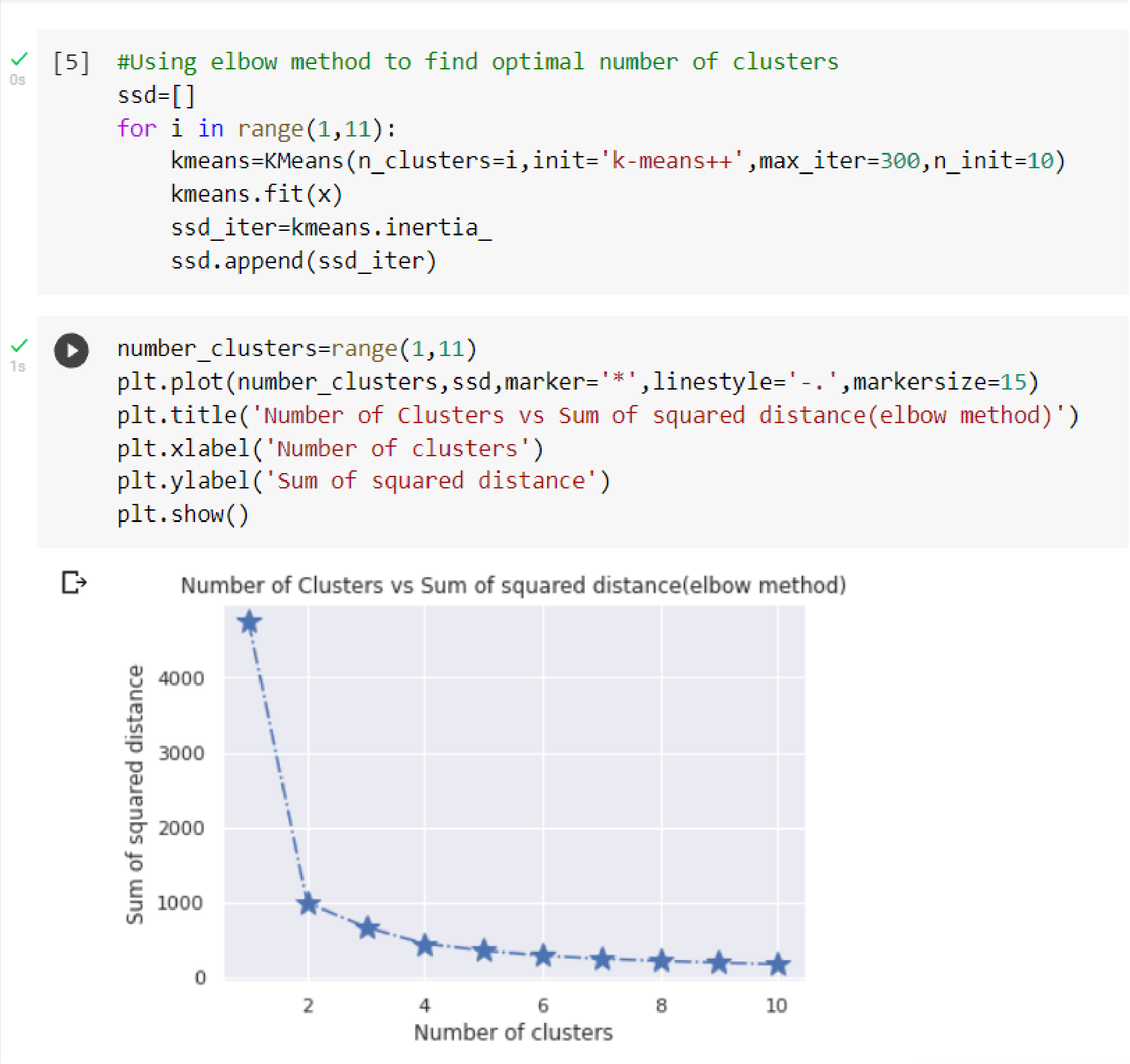
	latd	longd
0	34.576111	-117.432778
1	34.153333	-118.761667
2	37.756111	-122.274444
3	37.886944	-122.297778
4	34.081944	-118.135000



```
#Plotting scatter plot for raw data of longitude and latitude
plt.scatter(data['longd'],data['latd'])
plt.ylabel('Latitude')
plt.xlabel('Longitude')
plt.title('California cities latitude and longitude')
plt.show()
```

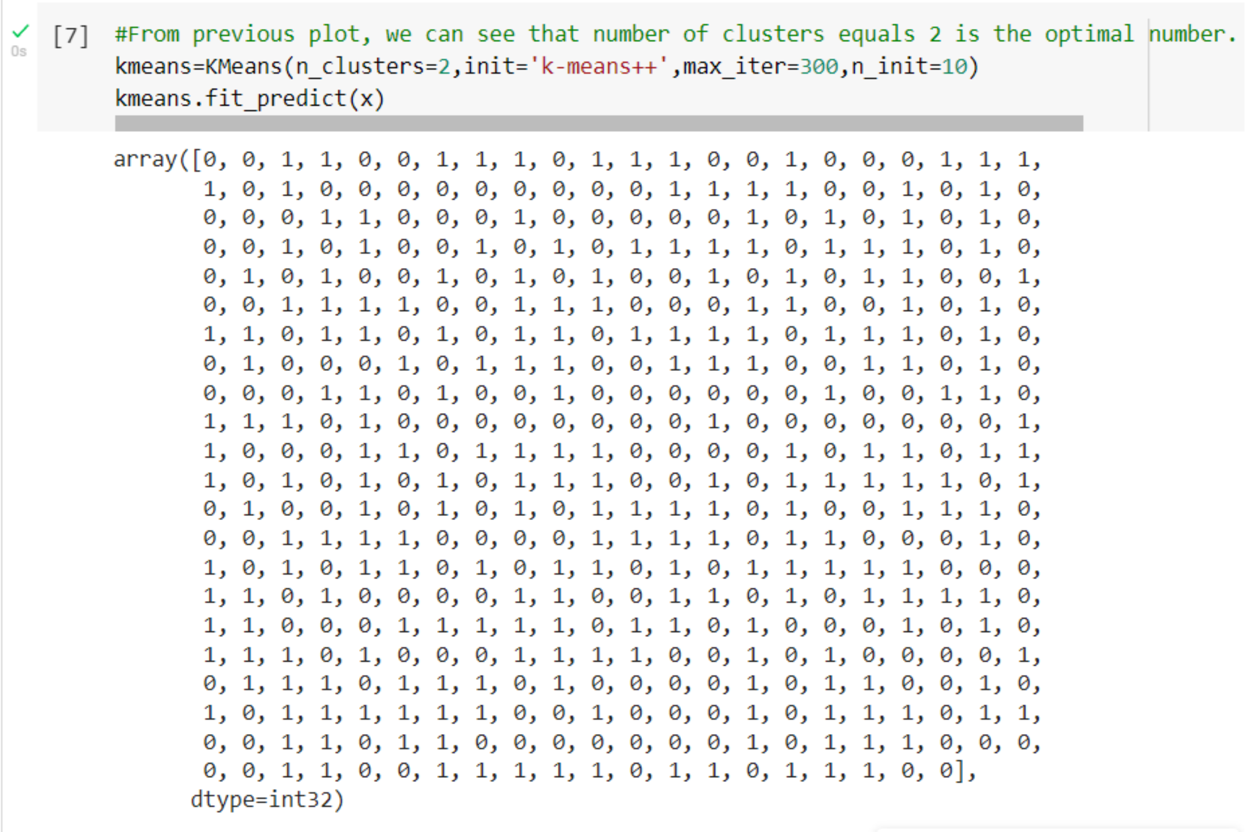
Elbow method

The elbow method helps to choose the optimum value of ‘k’ (number of clusters) by fitting the model with a range of values of ‘k’. We calculate the sum of squared distance(SSD) of each object from its cluster center, and from that, we find the number until which sharp changes in SSD are occurring. This point seems like an elbow and is the required number of k we want.



```
[5] #Using elbow method to find optimal number of clusters
    ssd=[]
    for i in range(1,11):
        kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10)
        kmeans.fit(x)
        ssd_iter=kmeans.inertia_
        ssd.append(ssd_iter)
```

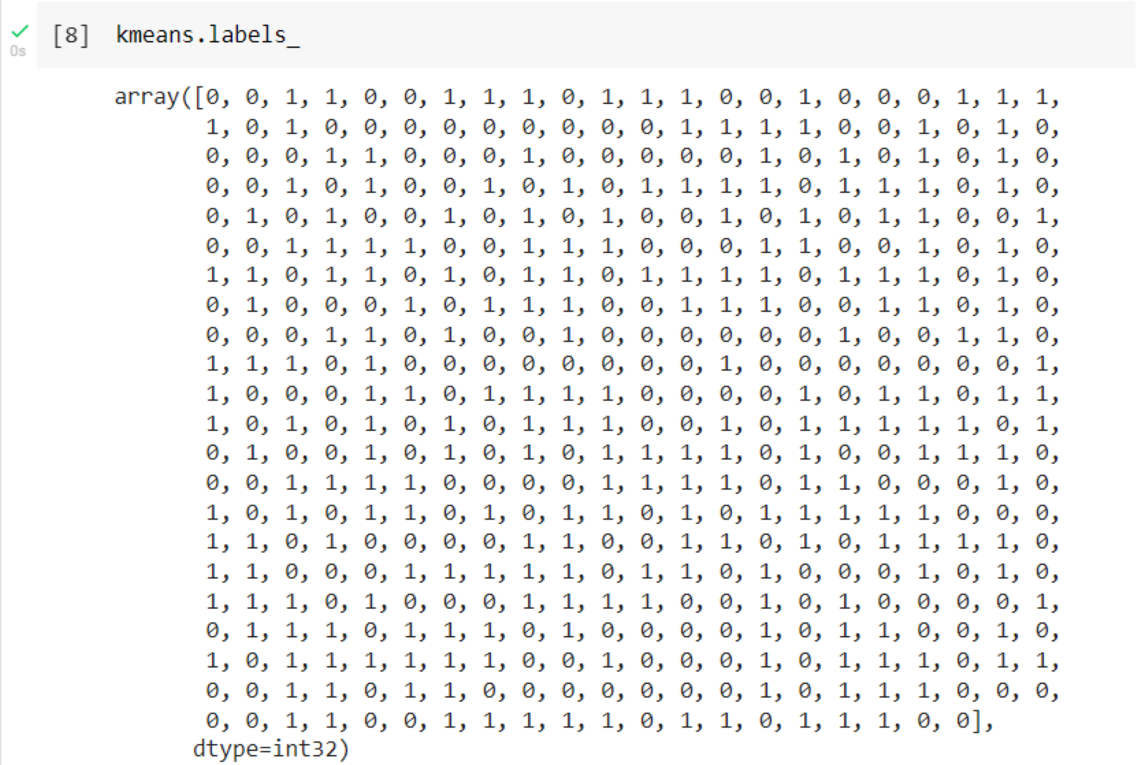
```
number_clusters=range(1,11)
plt.plot(number_clusters,ssd,marker='*',linestyle='-.',markersize=15)
plt.title('Number of Clusters vs Sum of squared distance(elbow method)')
plt.xlabel('Number of clusters')
plt.ylabel('Sum of squared distance')
plt.show()
```



[7]

```
#From previous plot, we can see that number of clusters equals 2 is the optimal number.
kmeans=KMeans(n_clusters=2,init='k-means++',max_iter=300,n_init=10)
kmeans.fit_predict(x)
```

```
array([0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0],
      dtype=int32)
```



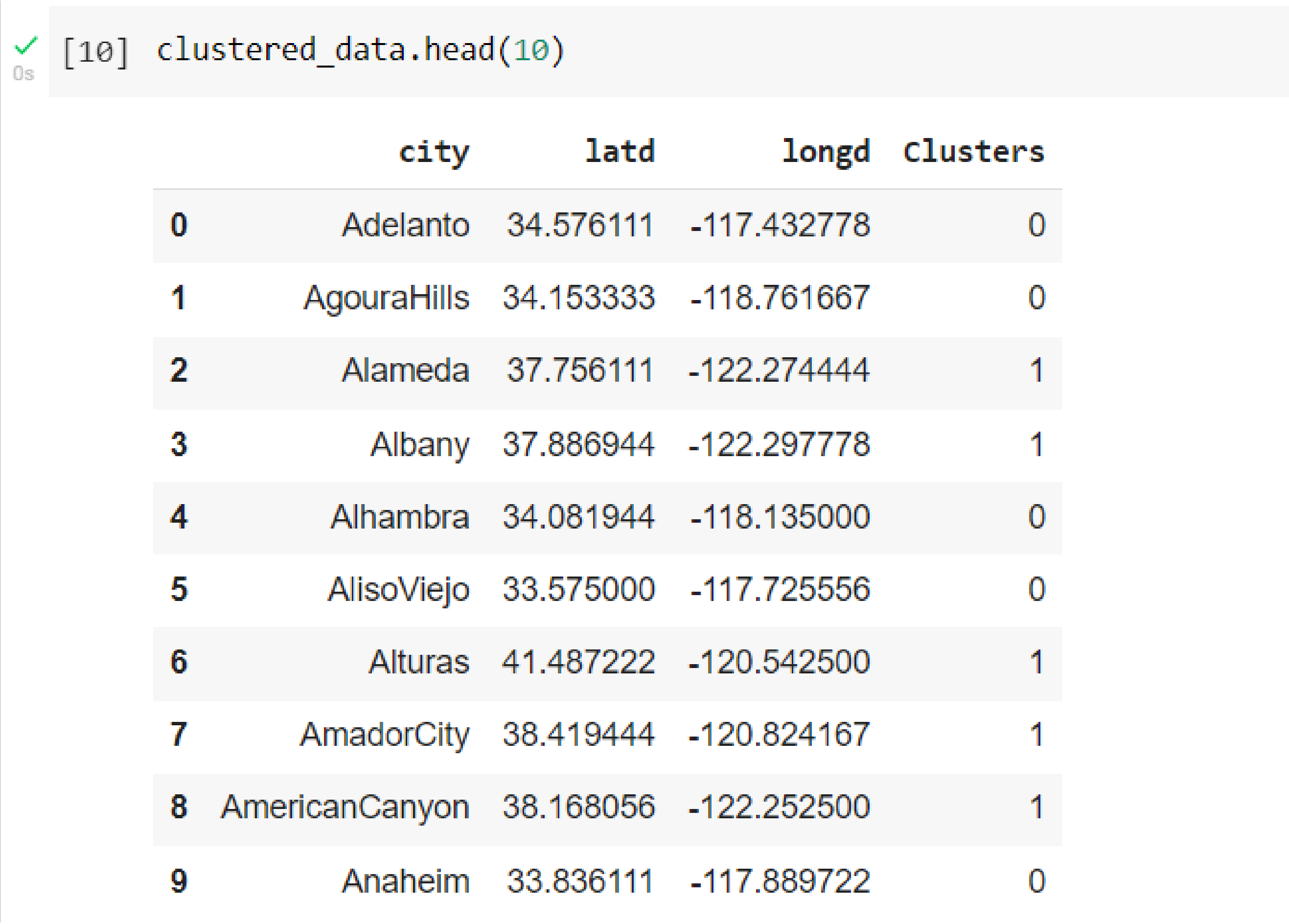
[8]

```
kmeans.labels_
```

```
array([0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0],
      dtype=int32)
```

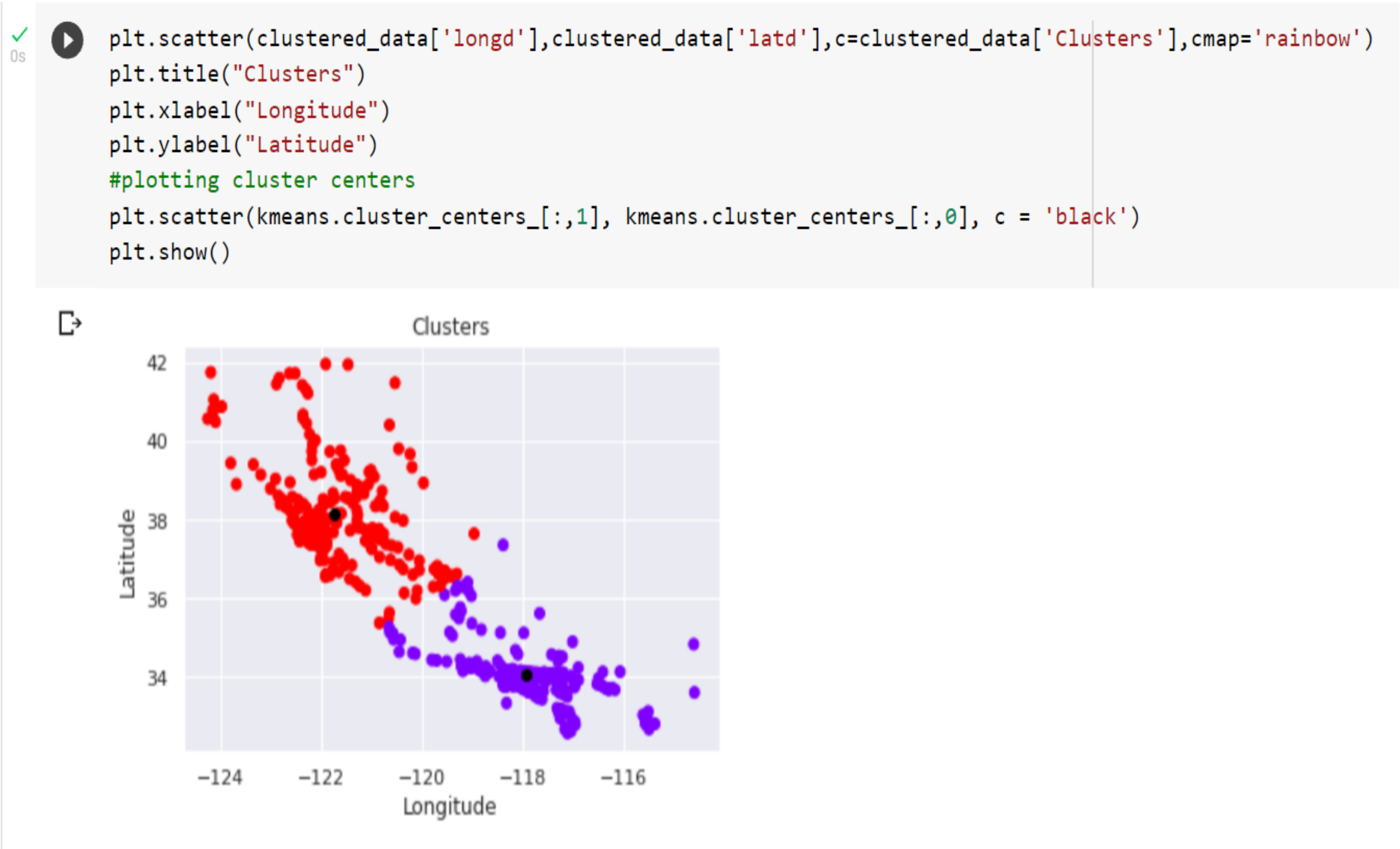
Using KMeans() to create model

Clustered Data after assigning to Clusters



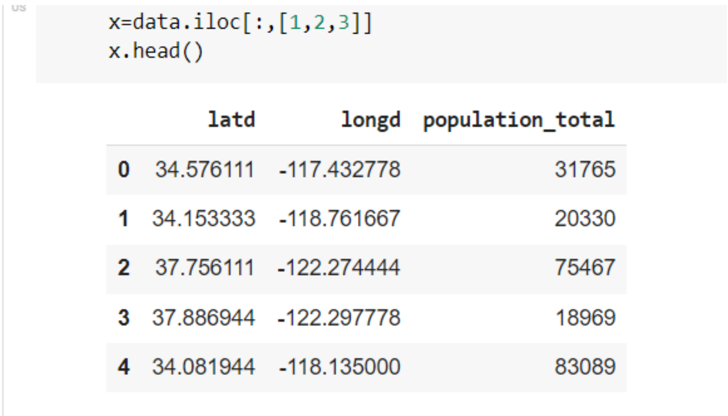
```
[10] clustered_data.head(10)
```

	city	latd	longd	Clusters
0	Adelanto	34.576111	-117.432778	0
1	AgouraHills	34.153333	-118.761667	0
2	Alameda	37.756111	-122.274444	1
3	Albany	37.886944	-122.297778	1
4	Alhambra	34.081944	-118.135000	0
5	AlisoViejo	33.575000	-117.725556	0
6	Alturas	41.487222	-120.542500	1
7	AmadorCity	38.419444	-120.824167	1
8	AmericanCanyon	38.168056	-122.252500	1
9	Anaheim	33.836111	-117.889722	0



```
plt.scatter(clustered_data['longd'],clustered_data['latd'],c=clustered_data['Clusters'],cmap='rainbow')
plt.title("Clusters")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
#plotting cluster centers
plt.scatter(kmeans.cluster_centers_[:,1], kmeans.cluster_centers_[:,0], c = 'black')
plt.show()
```

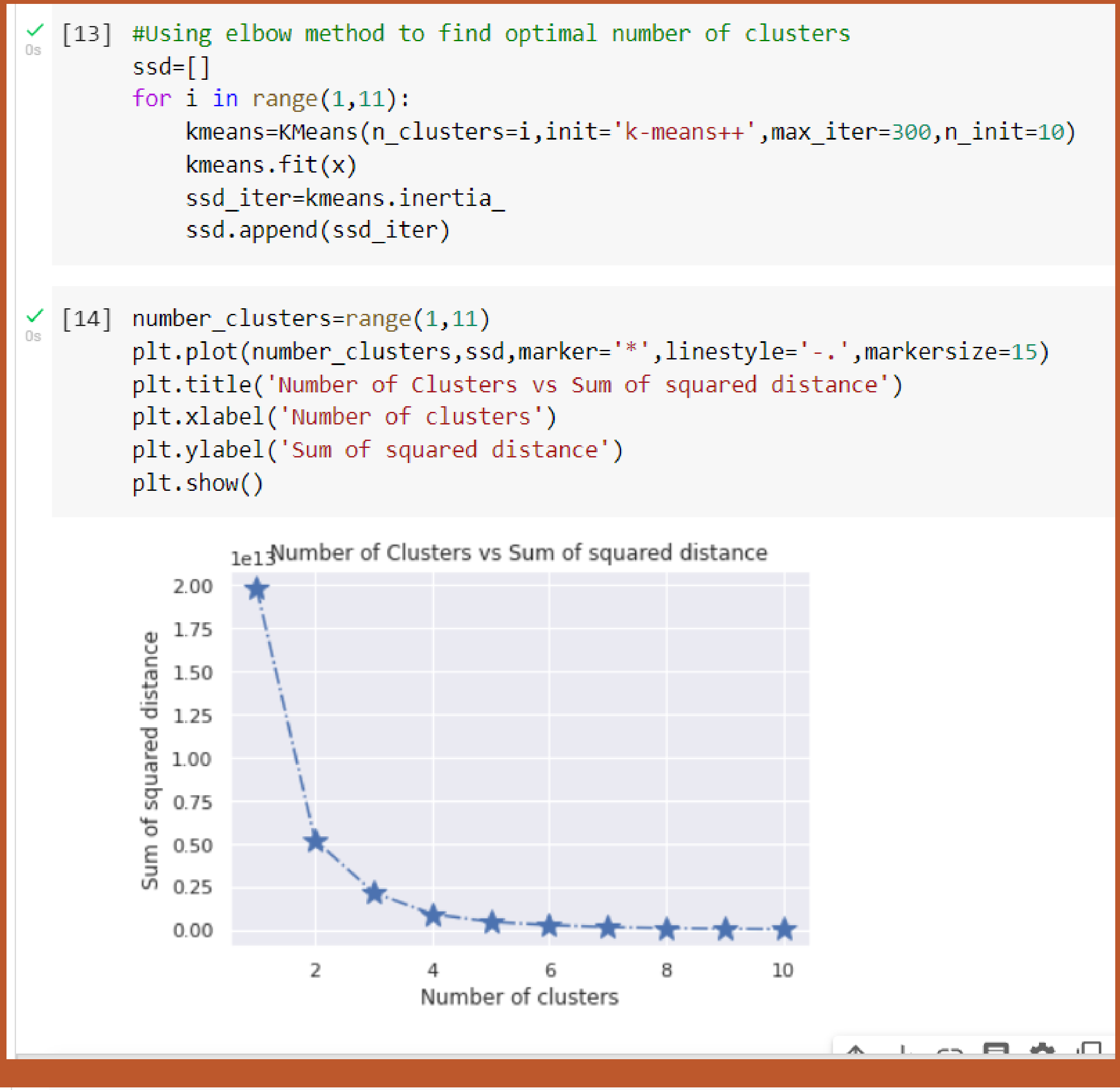
Plot scatter plot for clustered data



```
x=data.iloc[:,[1,2,3]]
x.head()
```

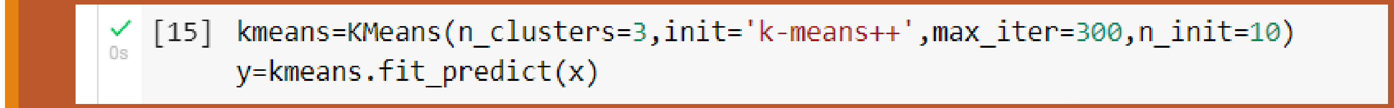
	latd	longd	population_total
0	34.576111	-117.432778	31765
1	34.153333	-118.761667	20330
2	37.756111	-122.274444	75467
3	37.886944	-122.297778	18969
4	34.081944	-118.135000	83089

Cluster data on basis of 2 or more variable



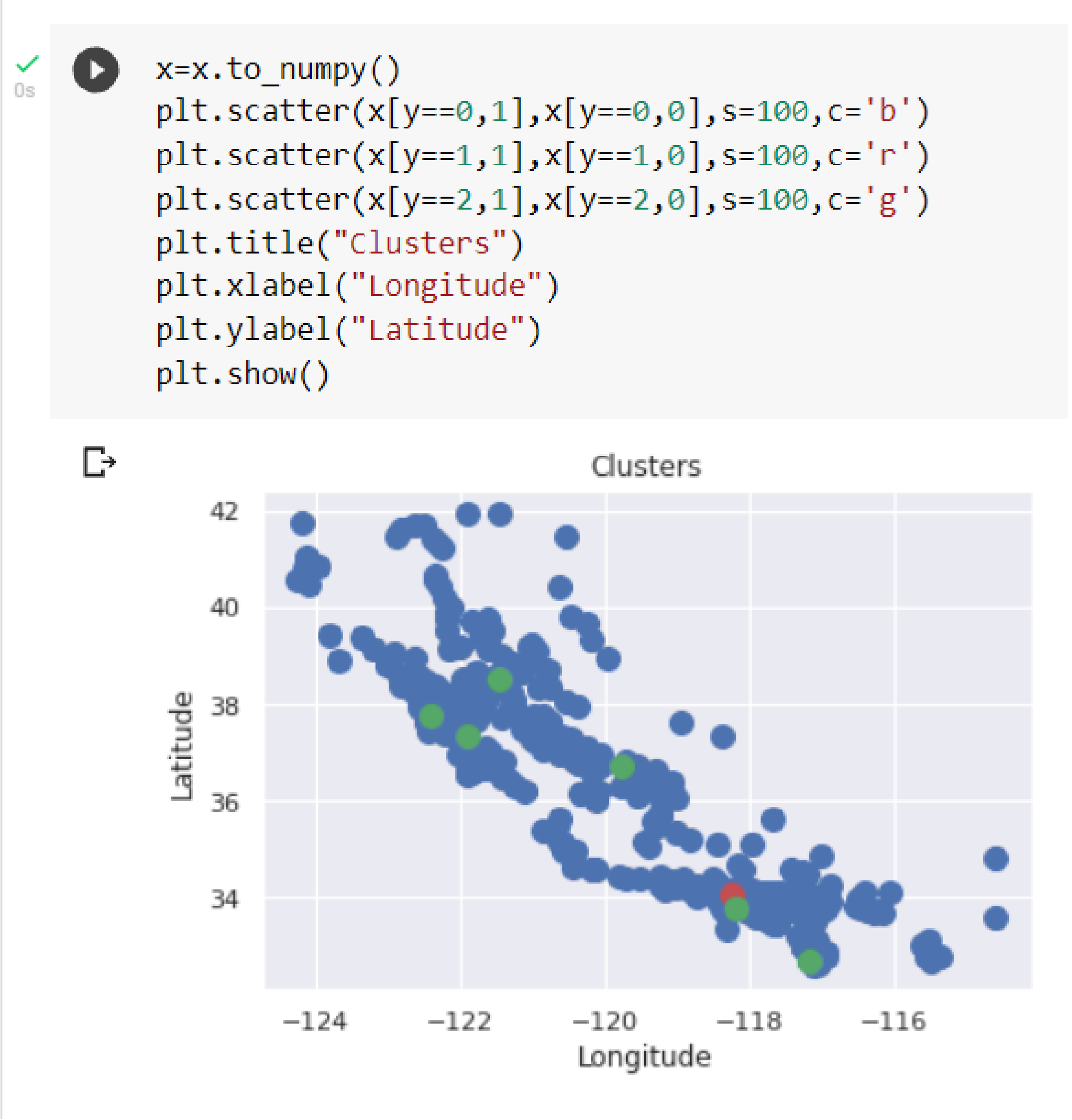
```
[13] #Using elbow method to find optimal number of clusters
     ssd=[]
     for i in range(1,11):
         kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10)
         kmeans.fit(x)
         ssd_iter=kmeans.inertia_
         ssd.append(ssd_iter)
```

```
[14] number_clusters=range(1,11)
     plt.plot(number_clusters,ssd,marker='*',linestyle='-.',markersize=15)
     plt.title('Number of Clusters vs Sum of squared distance')
     plt.xlabel('Number of clusters')
     plt.ylabel('Sum of squared distance')
     plt.show()
```



```
[15] kmeans=KMeans(n_clusters=3,init='k-means++',max_iter=300,n_init=10)
     y=kmeans.fit_predict(x)
```

Plot scatter plot for clustered data



```
x=x.to_numpy()
plt.scatter(x[y==0,1],x[y==0,0],s=100,c='b')
plt.scatter(x[y==1,1],x[y==1,0],s=100,c='r')
plt.scatter(x[y==2,1],x[y==2,0],s=100,c='g')
plt.title("Clusters")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```

Applications of K means clustering

Firstly, the courier company example for deciding where to open delivery center :)

It is used in the clustering of documents to identify the compatible documents in the same place.

It is helpful in the business sector for recognizing the portions of purchases made by customers, also to cluster movements on apps and website.

Clustering forms a backbone of search engines. When a search is performed, the search results need to be grouped, and the search engines very often use clustering to do this.

There are many more applications. You can always refer internet to know more

Advantages and Disadvantages of K Means clustering

ADVANTAGES

- It is very smooth in terms of interpretation and resolution.
- It can work on unlabeled numerical data.
- For a large number of variables present in the dataset, K-means operates quicker than other types of clustering.
- Uncomplicated to understand and yields the best outcomes when datasets are well distinctive (thoroughly separated) from each other.

DISADVANTAGES

- Sometimes, it is quite tough to forecast the number of clusters, or the value of k.
- K means is sensitive to outliers. Presence of outliers highly affect model.
- It is not directly applicable to categorical data since only operatable when mean is provided

Thankyou